Evaluation of MPSoC Operating Systems

Soler-Delgado David *1, Antonio-Torres David 2

Department of Electronics, Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Puebla. Puebla, México

1A01091632@itesm.mx; 2davidant@itesm.mx

Abstract

The development of Multi-Processors Systems on Chip (MPSoC) has increased recently due to their high processing speed and low-power consumption. Software implementation of these systems rely on parallel computing, a different programming paradigm which has not been fully adopted by most of the developers. Real-Time Operating Systems (RTOS) is one of the most important pieces in an embedded system. Unfortunately, some RTOSs have been mostly released for single processor embedded systems and there are just a few research-purposes multiprocessor operating systems. This paper explores some alternatives reported in the literature to the implementation of MPSoC operating systems. One of the most promising commercial MPSoC RTOS called FreeRTOS xcore is introduced and described. The characteristics of this MPSoC RTOS are contrasted with those of some other available RTOSs, concluding that FreeRTOS xcore is a suitable operating system for mid-range multiprocessor systems, where easy and fast implementation is a crucial constraint.

Keywords

FPGA; Multiprocessor System on a Chip; Real-time Operating Systems; FreeRTOS.

Introduction

Since their arrival to the commercial market, embedded systems have exponentially increased their importance for people's life. Nowadays, their easiness of implementation and high performance make them suitable solutions which can be exploited in many fields such as, digital signal processing and ubiquitous computing. However, as technology advances, it demands higher processing performance which cannot be achieved with a single-processor embedded system. Therefore, Multiprocessor Systems on a Chip (MPSoC) have received special attention, reaching the point where VLSI research has made possible to embed hundreds of cores into a single chip. Nevertheless, the increase in the number of processors has changed the typical programming paradigm making it more complex. Initially, parallel computing had been proposed for high-end multiprocessors systems, but today, there has been a lot of attention paid to multicore systems because of all their advantages such as increase on the speed of processing and reduction on the power consumption, features that are crucial in most of the current targeted applications.

An important component of any computational system is the operating system (OS), which is in charge of the task management, scheduling processes, memory allocation and peripheral control. For embedded systems, there is an OS variant called real-time operating system (RTOS). This kind of operating systems must comply with some constraints such as small memory footprint, high performance in worst-case situations and portability over different architectures. With the advent of MPSoC, all the RTOSs have become partially useless because their main features are not planned to work in parallel processing environments. Multi-processor symmetric operating systems have been proposed as a solution, which employ a set of single core operating systems working in a multiprocessor system environment. However, there are many conflicts that make this implementation not as reliable as expected. There have been some research-purpose MPSoC operating systems which offer high performance, but some important issues that affect its public release are lack of scalability, complex implementation procedures and poor throughout in processor load balance. Furthermore, source code is not publicly distributed, so community improvements cannot be achieved.

FreeRTOS is a commercial real-time operating system that offers free license (Barry, R. FreeRTOS) and public community is allowed to modify the source code provided that it is redistributed. In September of 2011, James Mistry developed a multi-processor operating system based on the FreeRTOS kernel called FreeRTOS xcore (Mistry, J. FreeRTOS and Multicore, 2011). This operating system proposes a quick solution with acceptable performance for mid-range multiprocessor embedded systems.

The paper is organized as follows. Section 2 resumes

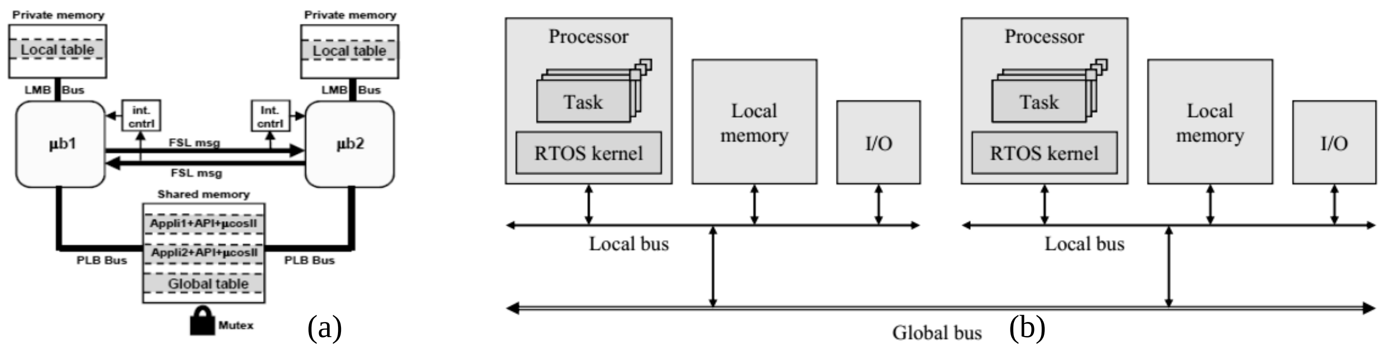


FIG. 1 (a) HARDWARE ARCHITECTURE IMPLEMENTED BY GANTEL (L. GANTEL, MULTIPROCESSOR TASK MIGRATION IMPLEMENTATION IN A RECONFIGURABLE PLATFORM, 2009), (b) HARDWARE ARCHITECTURE IMPLEMENTED BY (TOMIYAMA H. TOMIYAMA REAL-TIME OPERATING SYSTEMS, 2008)

the state of art on the development and performance of research-purpose MPSoC operating systems. Section 3 introduces the minimum requirements needed for the implementation of FreeRTOS xcore based on our personal experiments using a Xilinx Spartan 3E starter board. In addition, section 3 evaluates and compares the characteristics of this implementation with those of some other systems that have been reported in the literature. Section 4 concludes that, despite FreeRTOS xcore still has some features to be optimized, it represents an option for rapid implementations of multiprocessor embedded system. Features such as easy software programming procedures as well as high performance compared with a single processor RTOS make it a good option for novel programmers and developers in parallel computing

State of Art

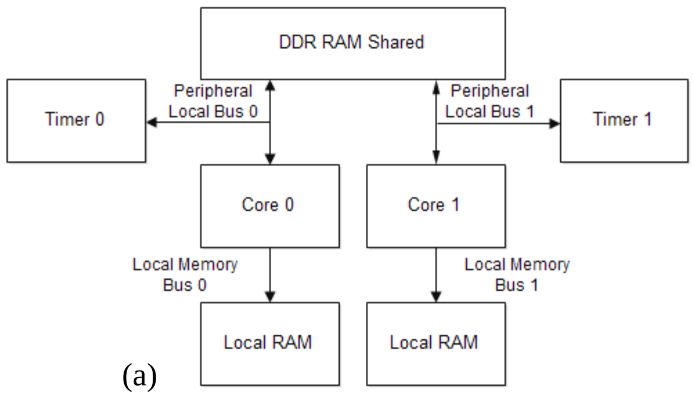
Architectures and Topologies

Multiprocessor systems started to being developed for high-end processing systems. However, the evolution in technology demands more powerful processing systems. On the multi-processor embedded system, there are two types of architectures, symmetric and asymmetric systems.

Symmetric multiprocessing system is the architecture where two or more homogeneous processors run the same instance of the RTOS. All the processing entities are connected into a single shared main memory, mainly used as an interconnection path. The peripherals on the system can be also shared by the processors and their access is gained using mutual exclusion techniques implemented by software or hardware. Symmetric systems offer the easiest software programming techniques because the operating system is in charge of the dynamic allocation of tasks in order to balance the loads of the processors. However, assigning the balance load to the operating systems deteriorates the worst-case performance, a critical parameter that has to be complied with by RTOSs.

Asymmetric MPSoC is an alternative to symmetric systems. In this case, processors can be homogenous or heterogeneous. Each processor has its own local memory in which its own instance of the RTOS is located. Networks on Chip (NoC) are implemented to allow having access to external processor memories (i.e. inter-processor communication). Even though the implementation of these networks increases the complexity of the hardware and software design, these systems are widely used in embedded systems where task allocation can be fixed at the design time. Consequently, this improves the worst-case performance and a task can be optimized to be executed on a specific processor by means of power consumption and processing speed. On the other hand, load balance is limited due to the small throughput of the processor interconnection, result of the sophisticated protocol followed by the NoC. Moreover, programming complexity is increased since inter-processor message-passing is not straightforward and APIs cannot be generalized since a special instance of an operating system must be done for every processor. (Oshana, R. DSP for Embedded and Real-Time Systems, 2012. H. Tomiyama Real-Time Operating Systems, 2008).

There have been few implementations of multi-processor systems reported in the literature. Nevertheless, those systems have been focused on seeking for improvements in some features such as task migration, load balancing or processing power, but just a few of them have been planned to be



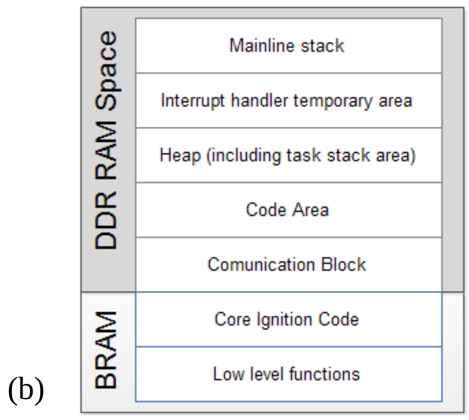


FIG. 2 (a) HARDWARE ARCHITECTURE USED FOR FREERTOS XCORE IMPLEMENTATION (b) MEMORY MAP MODEL OF SHARED AND LOCAL SPACES (MISTRY, J. FREERTOS AND MULTICORE, 2011)

released as a complete solution for public use. The next sub-sections summarize two of the most complete MPSoC RTOS solutions for their further comparison with a multiprocessor modified version of FreeRTOS.

SMP under μCOS-II

On 2009, Gantel et al. developed a variant of μCOS-II (L. Gantel, Multiprocessor Task Migration Implementation in a Reconfigurable Platform, 2009) under a Xilinx Virtex-5 FPGA using MicroBlaze processors with the main purpose of studying task migration between single processor instances of the RTOS, emulating a full symmetric multi-processor system.

The global hardware architecture is shown in Figure 1a. Each processor contains its own private memory, which contains local tables that point to task and services that are to be run. Two Fast Simplex Links (FSL) have been placed to make inter-processor communication emulating a point to point NoC. This communication is used to interact with private memories of external processors (i.e. inter-process communication).

A shared-memory scheme is mainly used to do task migration, which has been implemented by following replication strategy. This means that a replica of the task is located in every private memory of the system and shared memory is just the path used to pass context switching between the processors. Although replication strategy seems to provide a fast way to transfer tasks between processors, the complex programming procedure behind it makes its release difficult as a commercial RTOS. Further, scalability of the system is proposed but not straightforward and it also requires of more dedicated algorithms.

μITRON

One of the most complete developments of an MPSoC operating system has been developed by Tomiyama et al (H. Tomiyama Real-Time Operating Systems, 2008) through their Kernel called TOPPERS/FMDP, which follows μITRON specification. μITRON is one of the most popular RTOS in many Asia and Pacific countries (TRON Association).

Opposite to Gantel implementation, μITRON proposes an asymmetric architecture targeted for mid-range real-time systems, which lack large memory capacity. Figure 1b shows the global architecture proposed for the system. Each processor has a local bus in which all the peripherals as well as the I/O interface are connected. At the same time, all the processors have access to a global bus.

Unfortunately, task migration is not allowed because of the lack of an efficient path to transfer the context, leaving the use of the global bus just for inter-task and inter-processor communication. Despite their limited capabilities, the positive side is reflected on the software programming where the complexity of code is substantially reduced (i.e. task implementation is really simple) as well as on the improvements of the worst-case performance.

Evaluation and Implementation of FreeRTOS xcore

Requirements

FreeRTOS xcore , a symmetric middle-range MPSoC operating system released for MicroBlaze processor for versions up to 8.00b (or for EDK versions up to 13.1) (Xilinx. MicroBlaze Processor Reference Guide, (2011), was developed by James Mistry in 2011 as an extension of the original FreeRTOS kernel (Mistry, J. FreeRTOS and Multicore, 2011). The minimal implementation diagram used for two homogenous processors is shown in Figure 2a. Each processor has its own local memory (16 KB), which contains individual code of each processor (i.e. the Board Support Package), a personal timer used for the pre-emptive scheduler in order to specify the time for task switching and, finally, a shared DDR RAM connected through a MultiPort Memory Controller (MPMC) (Xilinx. LogiCORE IP Multi-Port Memory., 2011). The MPMC is the peripheral that allows both processors to read and write simultaneously from the DDR RAM. In fact, this controller is the heart of the MPSoC, and, on the literature, there is none proposal that uses this peripheral on the same way as has been proposed to be used here.

Although MicroBlaze Debug Module (MDM) is not shown on the figure, it is also included on the system because this peripheral is used to write the source code into the DDR RAM and local memories once the system has already been built.

In Figure 2a, processor 0 is labeled as master core, while the remaining processors are labeled as slave cores. Both types of processors work in the same way once the system has been initialized. During initialization stage, master core allocates the shared memory, starts the operating system, creates the tasks for all the processors, runs the scheduler, creates the entry pointer address to it, and notifies all the slave processors that tasks are ready to be executed. Consequently, each slave processor waits for the notification of the master processor, then jumps to address of the scheduler, and picks up its corresponding task.

This simplistic model is the main advantage of the FreeRTOS xcore, as the code implemented on the master processor is the only one different in the system. Slave codes are just in charge of processor and pointer initialization (i.e. they run the code located on its local space, Figure 2b), while master code is larger because all the tasks for any processor are specified at once.

Scheduler and Load Balance

One of the main features provided by FreeRTOS xcore is the dynamic load balance which is possible due to the modification of the Task Control Board (TCB) of the original kernel. A special field has been added to the TCB called core affinity that contains the ID of the core in which the task is assigned to or it contains 0 if the task does not belong to none of them, indicating that this task can be processed by any of the present processors. Master and slave processors are running at the same frequency but with any synchronization system between them. When one of the processors is interrupted by its timer, it performs context switching and seeks for the next non-started task available that belongs to this processor. Once the task has been selected from the TCB, the processor takes the pointer to the function and starts working on it. Again, when the tick occurs, the processor releases the pointer and modifies the TCB to notify the other processors that the task is available. If task affinity is specified for both processors, then dynamic load balance is performed automatically by the scheduler.

Inter-processor Communication and Sychronization

Although the system is similar to the one implemented by Gantel, the FSL is missing. Additionally, there is no global bus that can be used as a communication path like Tomiyana implementation. Therefore, inter-process communication is performed through the shared memory using the communication block area located over the DDR RAM space. To provide safe transfer, shared memory has to be segmented using the linker script of the master processor. The resulting memory map seen by all the cores is shown in **Figure 2b**. The explanation of functionality of each block is out of the scope of this section.

Inter-process synchronization procedures are performed using mutual exclusion by software using Peterson's algorithm (Mistry, J. FreeRTOS and Multicore, 2011). However, there is an important hardware feature of MicroBlaze processor that makes possible the inclusion of a processor into critical code sections without being interrupted by others. Once one processor enters into one of these sections, it notifies the MPMC that the processor cannot be interrupted and then a memory barrier is created for the rest of the processors that attempt to get into the critical region. Without this feature, it would be impossible to manage the race conditions in MPSoC, leading the system to deadlock.

Implementation and Evaluation

In this section, the issues during the implementations of FreeRTOS xcore are pointed out. Both the advantages and disadvantages of this RTOS with those previously described in section 2 are contrasted.

An MPSoC of two processors has been implemented on a Xilinx Spartan-3E starter board using the lightest version of MicroBlaze 8.00b (minimum version which supports memory barrier) using Xilinx Platform Studio 13.1. One of the main limitations during the implementation is the number of BRAM blocks and available LUTs.

As previously stated, MPMC is the heart of the system, which is the most controversial issue, resulting from the fact that MPMC occupies the largest space on the FPGA because the emulation of simultaneous read/write multiport controller needs some cache device that must be implemented using BRAMs or Shift Register LUTs (SRL). MPMC gives many options to configure the cache on each port, so it is possible to balance the number of LUTs and BRAM used; however, the entire peripheral is really costly. To achieve an estimate on the footprint of the MPMC, the next discussion calculates the number of BRAMs and LUTs needed. For the cache of one MPMC port of 32-bit wide (MicroBlaze PLB standard width) 4 BRAM are used plus an extra BRAM, which is intended for internal purposes of the MPMC. Therefore, for a dual processor system, the MPMC needs 9 BRAMs just to share the memory space. Furthermore, each processor needs 16 KB of local RAM to hold local functionalities. This space is translated into 8 BRAM blocks. With a total of 21 BRAMs, the dual processor system does not fit into the Spartan 3E (XC3S500E) (Xilinx. DS312 Spartan-3E FPGA Family, 2012). It is possible to exploit the capabilities of MPMC using SRLs. This feature has made possible the implementation of the system in a mid-size device. Changing BRAM to SRL capabilities, each port needs around 1187 LUTs and 1478 flip-flops. This amount of logic elements used for MPMC is comparable with the amount of logic elements used by the lightest version of MicroBlaze. Implementation simplicity versus implementation area is one of the main trades off presented in this system. MPSoC RTOSs such as μITRON or SMP-μCOS have lower hardware constrains; however, the downside effects are shown on the software programming side.

For the system that has been implemented, after the hardware implementation, software programming is simple. Those developers familiar with FreeRTOS distribution will find that software implementation is almost straightforward, and that little modification of the original task instantiation procedure is needed. Therefore, it can be clearly seen that FreeRTOS offers enormous advantages of task migration compared with SMP μCOS-II, whose task migration procedure is complex. On the other hand, task is as simple as μITRON implementation but enhanced with load balance features.

Simple benchmarking tests using concurrent tasks have been executed by Mistry to demonstrate the higher processing power of FreeRTOS xcore. These tasks have been performed over short, long and very long, which runs without deadlock. Results show that double-processor FreeRTOS is around 1.88 times faster than single-processor instance. This processing speed can be improved by increasing the number of processors on the system, but this factor will have impact on the implementation area requiring higher-range FPGA systems. However, scalability procedure is easy, as all the slave processing instances run exactly the same code.

Conclusion and Future Work

There exist research-purpose MPSoC RTOSs which, under experimental conditions, offer acceptable performances. However, it is believed that there is a need for a commercial multiprocessor operating system that can be quickly instantiated and configured on FPGA systems.

In this paper, two MPSoC RTOS proposals found in literature and FreeRTOS xcore have been explored in order to contrast their features. It can be concluded that FreeRTOS xcore has demonstrated to be a suitable option as a Multiprocessor System on chip RTOS, offering easy task management as well as higher processing performance and longer scalability characteristics.

Despite the fact that FreeRTOS xcore offers better features than other systems, there are some aspects that have not been explored yet, such as power performance, processing speed and deadlock probability in terms of the number of processing entities available in the system. One of the two future research paths that have been devised is to explore those aspects in order to eliminate FreeRTOS xcore weaknesses. The second path is focused on the definition of an application that demands higher processing performance and exploits the processing power of this Multiprocessor RTOS.

REFERENCES

Barry, R.. FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems supporting 33 microcontroller architectures. www.freertos.org. Accessed February 28th, 2013

H. Tomiyama, S. H. IEEE . Real-Time Operating Systems for Multicore Embedded Systems.International SoC Design Conference. (2008). Jeju, South Korea:.

L. Gantel, S. L. IEEE. Multiprocessor Task Migration Implementation in a Reconfigurable Platform International Conference on Reconfigurable Computing and FPGAs. (2009) . Cancún, Mexico.

Mistry, J. FreeRTOS and Multicore. University of York. 2011.

Oshana, R. DSP for Embedded and Real-Time Systems. Newnes. Walham (2012).

TRON Association, http://www.tron.org/. Accessed February 28th, 2013

Xilinx. MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 13.1. ,2011. http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/mb_ref_guide.pdf. Accessed February 28th, 2013.

Xilinx. LogiCORE IP Multi-Port Memory, 2011 . http://www.xilinx.com/support/documentation/ipembedprocess_memoryinterface_mpmc.htm. Accessed February 28th, 2013.

Xilinx. DS312 Spartan-3E FPGA Family Datasheet ,2012. http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf. Accessed February 28th, 2013.